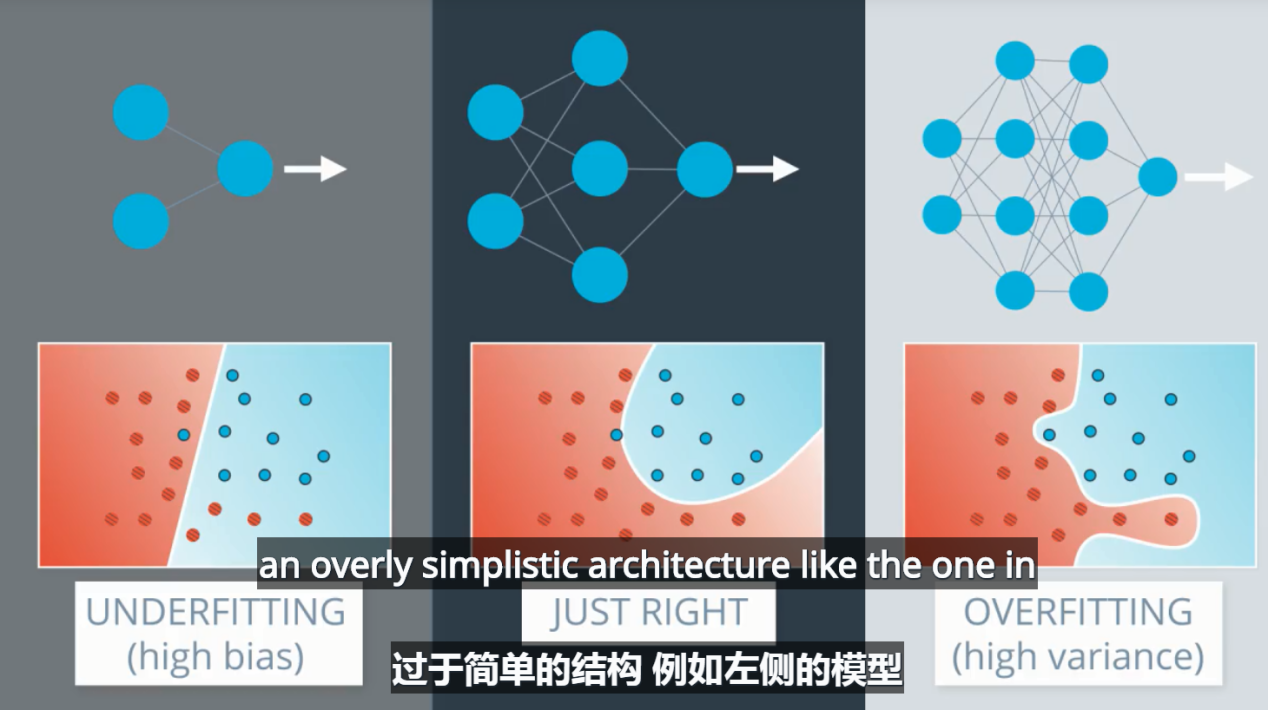
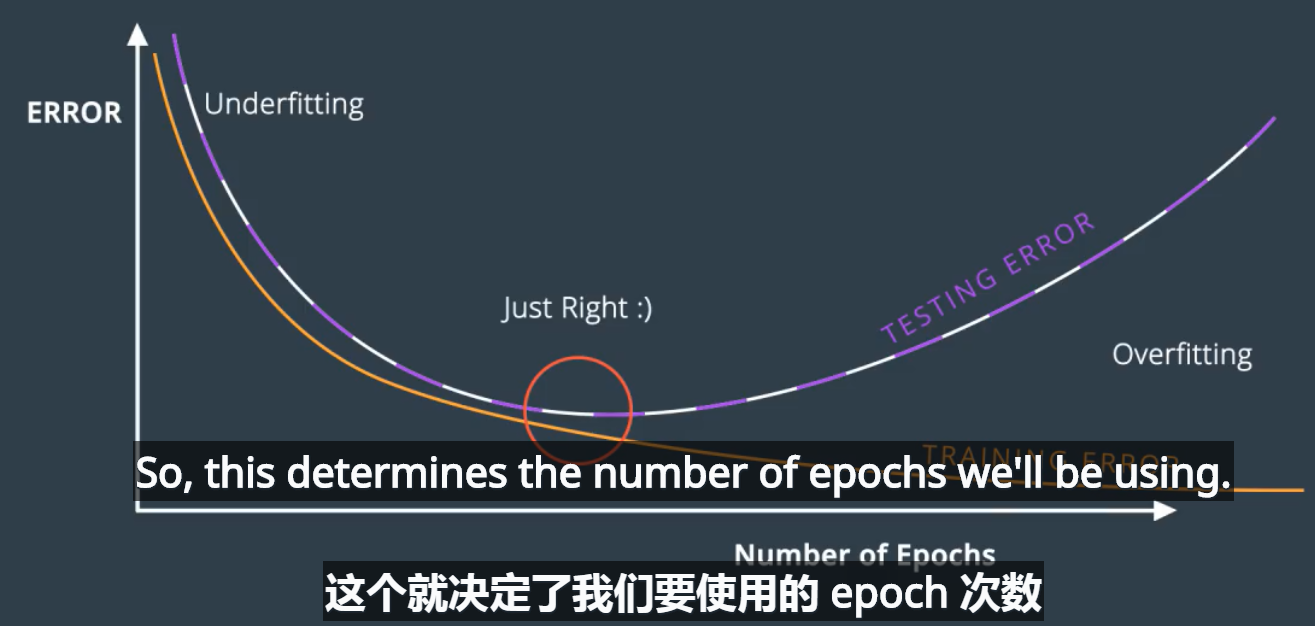
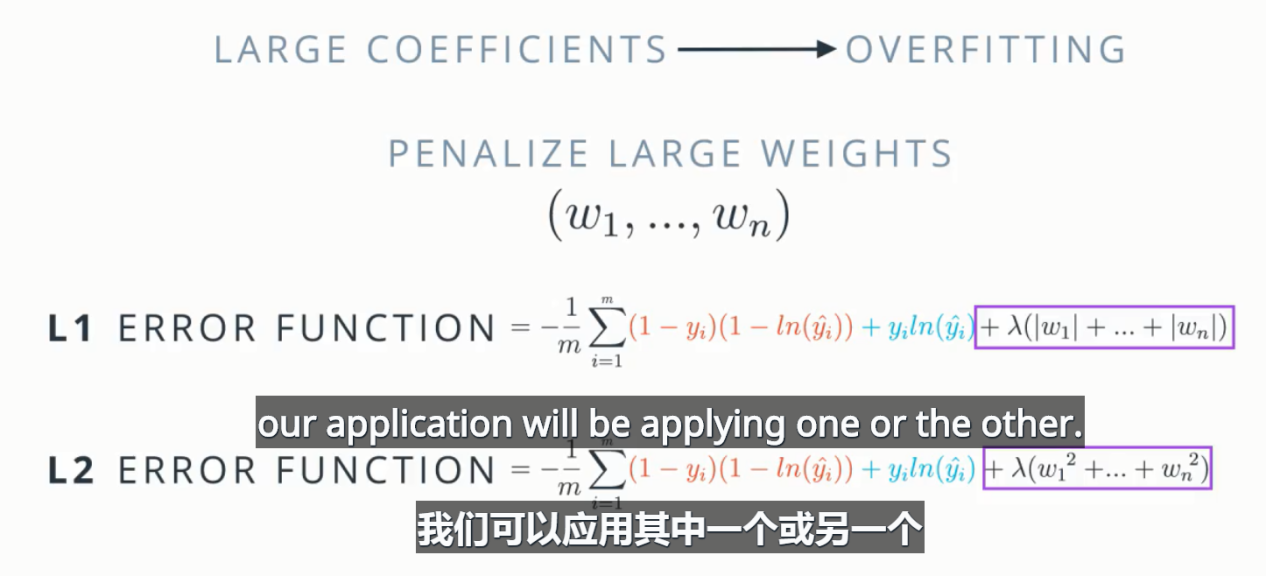
**1、过拟合和欠拟合**



**2、早期停止法来缓解欠拟合**

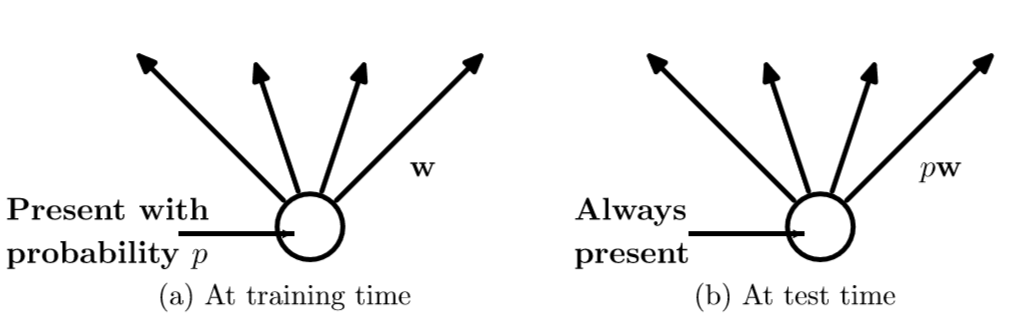


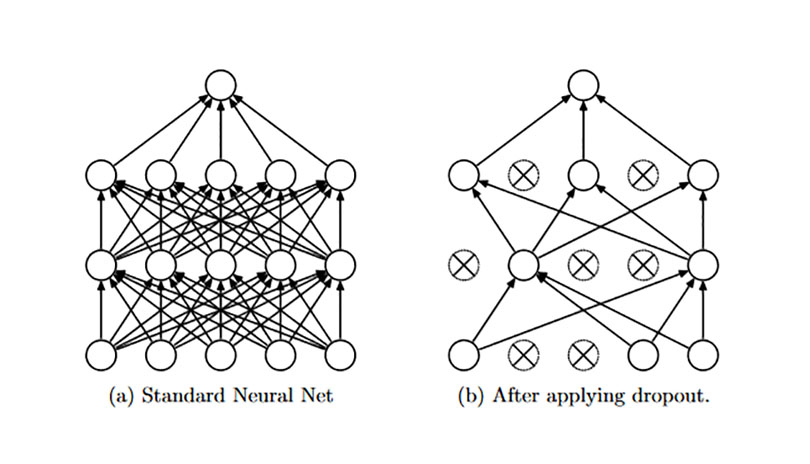
**3、L1L2正则化缓解过拟合**

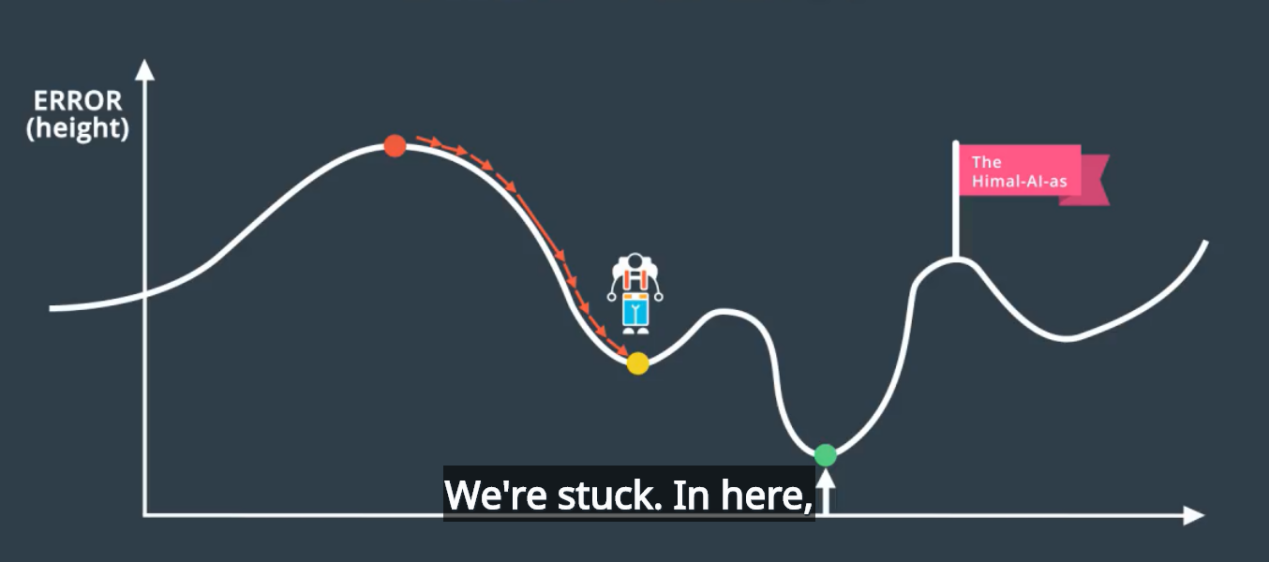


**4、dropout缓解过拟合**

Dropout是一种在深度学习环境中应用的正规化手段。它是这样运作的：在一次循环中我们先随机选择神经层中的一些单元并将其临时隐藏，然后再进行该次循环中神经网络的训练和优化过程。在下一次循环中，我们又将隐藏另外一些神经元，如此直至训练结束。  
在训练时，每个神经单元以概率p被保留(dropout丢弃率为1-p)；在测试阶段，每个神经单元都是存在的，权重参数w要乘以p，成为：pw。测试时需要乘上p的原因：考虑第一隐藏层的一个神经元在dropout之前的输出是x，那么dropout之后的期望值是E=px+(1−p)0 ，在测试时该神经元总是激活，为了保持同样的输出期望值并使下一层也得到同样的结果，需要调整x→px. 其中p是Bernoulli分布（0-1分布）中值为1的概率。示意图如下：

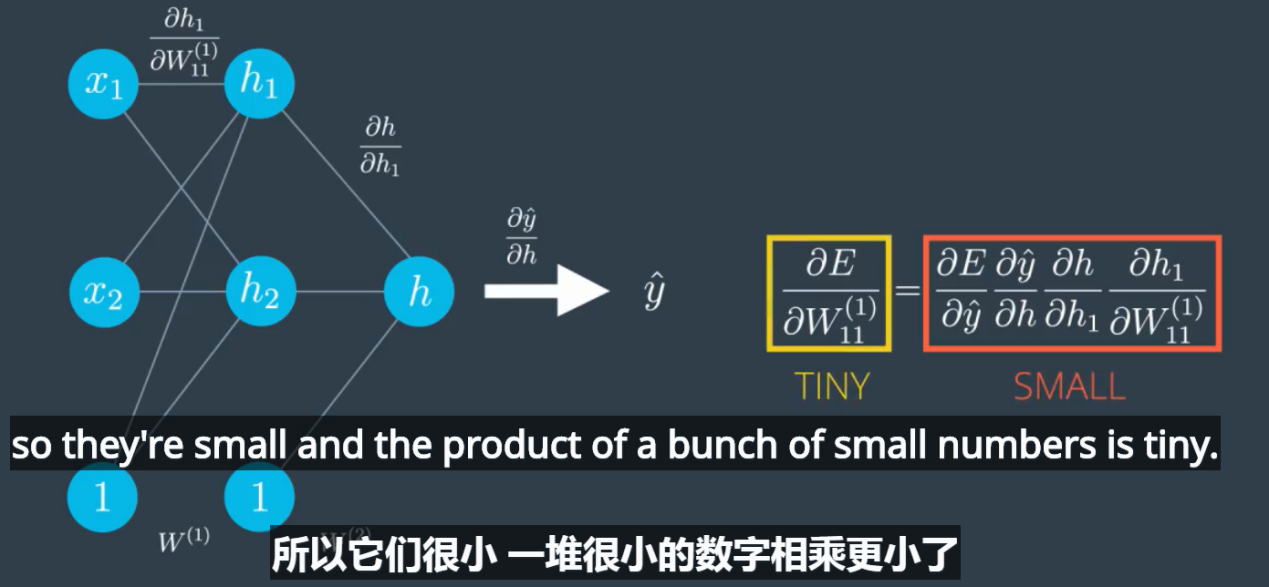




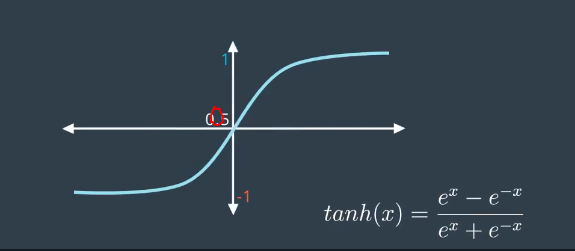


**5、多种激活函数**

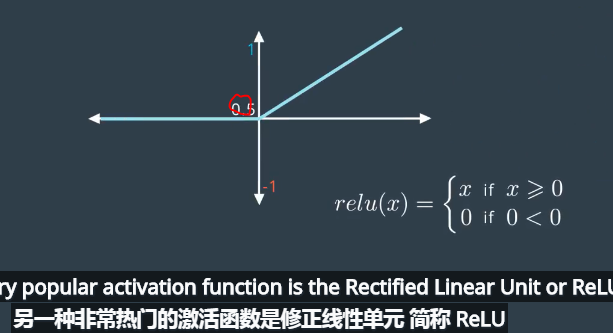
Sigmod函数存在两边梯度消失的问题，并且其梯度最大值也仅仅为1，当神经网络很深的时候，链式运算多项式子累积会导致算出的梯度很小，导致梯度消失。



双曲正切函数：



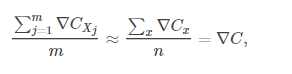
RELU函数:



1. **随机梯度下降和标准梯度下降**

根据前面的介绍，我们已经可以算出最小点在哪里了(理论上)，但是随机梯度下降还有一些问题，我在这里说两个：

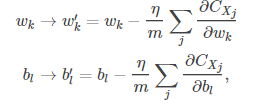
1. 容易陷入局部极小值，在前面的图中，我们只画出了一个全局极小值点，所以梯度下降可以直接找到最小点，但是在实际中，函数会有很多局部极小值，因此梯度下降可能会停止在局部极小值中，而不是全局极小值。
2. 计算量太大，注意到式(1)，我们计算所有输入数据计算一次，然后取平均，这样计算量太大了。这里采用随机梯度下降，即先将样本分组，对于每组1~m组数据，



两边互换一下，即可得：



应用到神经网络上，可以得到下式：



1. 随机梯度下降最大的缺点在于每次更新可能并不会按照正确的方向进行，因此可以带来优化波动(扰动)

随机梯度下降是通过每个样本来迭代更新一次，如果样本量很大的情况（例如几十万），那么可能只用其中几万条或者几千条的样本，就已经将theta迭代到最优解了，对比上面的批量梯度下降，迭代一次需要用到十几万训练样本，一次迭代不可能最优，如果迭代10次的话就需要遍历训练样本10次。但是，SGD伴随的一个问题是噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。

1. **解决局部最优解的其他方法**

**随机重新开始：**多个点重新开始递归，每个点均是局部最优，然后取代价函数最小的点，增大到达全局最优的概率

**动量梯度下降：即运动具有惯性，不容易被小的局部最优困住**

如果在峡谷地区(某些方向较另一些方向上陡峭得多，常见于局部极值点)，SGD会在这些地方附近振荡，从而导致收敛速度慢。这种情况下，动量(Momentum)便可以解决。动量在参数更新项中加上一次更新量(即动量项),即

